



RAPPORT TP APMAN

Vérificateur Orthographique

Binôme:

Mouhamadou SYLLA (syllamou)

Corentin TEISSIER (teissico)

2021/2022

Table des matières:

I°) État du logiciel

II°) Justifications des structures utilisées

III°) Tests effectués et Exemples d'exécutions

IV°) Analyse comparative des performances CPU et mémoire

I°) État du logiciel

On a mis en place 3 implémentations de structures de données:

- une séquentielle à l'aide de listes chaînées;
- une table de hachage;
- une structure arborescente avec un arbre préfixe.

L'ensemble des structures implémentées fonctionne. La compilation se fait sans erreurs ni warnings. Valgrind montre qu'il n'y a pas de fuites mémoires.

Le fonctionnement est aussi correct, les mots faux sont bien affichés comme faux et tout mot correct est identifié comme correct.

II°) Justifications des choix effectués

a) Choix de la liste chaînée

Parmi les implémentations qu'il fallait réaliser, on devait en faire une qui réalise une recherche séquentielle. En C, les structures qui permettent ce traitement sont les tableaux et les listes. Étant donné qu'un tableau a une taille inchangable déclarée lors de l'initialisation, on a décidé de mettre en place une liste chaînée.

Le programme de test utilisant cette méthode fonctionne comme suit:

D'abord on crée le dictionnaire: on crée une liste vide, on lit le fichier du dictionnaire FR.txt ligne par ligne à l'aide de fgets (chaque ligne correspond à un mot) et on crée un nouveau maillon pour chaque mot lu (le maillon contient le mot en chaîne de caractères) que l'on ajoute en tête de la liste et non en queue afin d'éviter de parcourir inutilement la liste.

Une fois cela de fait, on a décidé de ne pas afficher les erreurs directement à la console mais de les afficher avec leurs nombres d'occurrences. Pour cela, on a ajouté un champ nombre d'occurrences à la structure liste et on a créé une nouvelle liste appelée liste_erreurs qui stocke l'ensemble des erreurs trouvés. Ceci permet non seulement de pouvoir afficher les occurrences des

mots mais ça permet aussi d'éviter de toujours rechercher le mot dans la liste dictionnaire qui est très grande on recherche chaque mot dans la liste d'erreurs qui est "petite". Si le mot est présent dans cette liste, on incrémente juste son nombre. S'il n'y est pas, il y a deux possibilités: soit on ne l'avait pas rencontré, soit c'est un mot correct. On le recherche dans le dictionnaire et on agit en conséquence. Pour la recherche du mot, on récupère les lignes du texte une à une (avec fgets) et on récupère les différents mots de chaque ligne séparés par des séparateurs (avec la fonction strtok). Chaque mot est ensuite recherché dans les structures précisées précédemment grâce à des fonctions implémentées dans nos structures de données.

b) Choix de la table de hachage

Comme deuxième implémentation, on a choisi la table de hachage parce que c'est la structure qui est le plus souvent utilisée pour représenter un dictionnaire (les implémentations de dictionnaire dans d'autres langages tels que Python et Java reposent même sur le principe de la table de hachage). On a alors pris une fonction de hachage efficace qui permet d'éviter un très grand nombre de collisions tout en évitant que la table soit trop longue. Le fonctionnement de la fonction de test est similaire à celle de la liste chaînée avec l'utilisation d'une autre table de hachage qui stocke les erreurs. La table de hachage, qui stocke le dictionnaire, a été initialisée avec une taille de 100000 qui est environ le tiers de l'ensemble des mots dans le dictionnaire. Cette valeur nous semble être un bon choix étant donné que ce nombre permet d'éviter un nombre trop importants de collisions (3.56 en moyenne) tout en évitant d'utiliser une mémoire trop importante. Pour celle sur les erreurs, on a pris une taille de 200 qui permet aussi d'assurer les points précédemment mentionnés (on aura une moyenne de 15 collisions dans la table des erreurs mais cela n'est pas très important car on effectue beaucoup moins de recherche dans cette table).

c) Choix de l'arbre préfixe

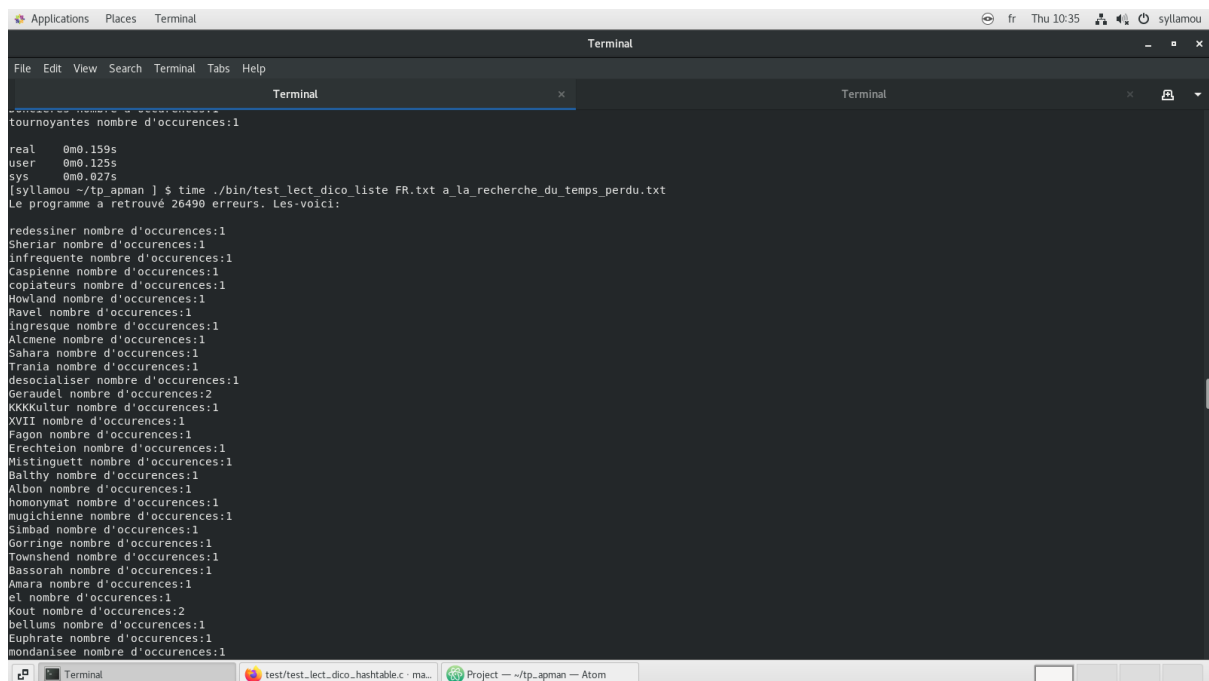
Il nous était recommandé de réaliser une implémentation arborescente, on a donc choisi de réaliser l'arbre préfixe qui est aussi bien adapté à la réalisation d'un dictionnaire. Cette fois-ci, on n'a pas repris le même type de fonctionnement que pour les deux précédentes structures. En effet, là où

parcourir une plus petite liste ou table de hachage avait un intérêt, il n'y a aucun avantage à parcourir un arbre d'erreurs plutôt que l'arbre du dictionnaire, les deux parcours étant exactement équivalents. On a donc décidé d'afficher les erreurs trouvés directement au niveau de la console.

III°) Tests effectués et exemples d'utilisation

Voici les résultats obtenus lors des différents test réalisés:

- Le test de la structure de la liste chaînée



```
Applications Places Terminal
Terminal
File Edit View Search Terminal Tabs Help
Terminal
tournoyantes nombre d'occurrences:1
real 0m0.159s
user 0m0.125s
sys 0m0.027s
[syllamou ~/tp_apman ] $ time ./bin/test_lect_dico_liste FR.txt a_la_recherche_du_temps_perdu.txt
Le programme a retrouvé 26490 erreurs. Les-voici:
redessiner nombre d'occurrences:1
Sheriar nombre d'occurrences:1
infrequente nombre d'occurrences:1
Caspienne nombre d'occurrences:1
copiateurs nombre d'occurrences:1
Howland nombre d'occurrences:1
Ravel nombre d'occurrences:1
Ingresque nombre d'occurrences:1
Alcmene nombre d'occurrences:1
Sahara nombre d'occurrences:1
Trania nombre d'occurrences:1
desocialiser nombre d'occurrences:1
Geraudel nombre d'occurrences:2
KKKKultur nombre d'occurrences:1
XVII nombre d'occurrences:1
Pagon nombre d'occurrences:1
Erechteion nombre d'occurrences:1
Mistinguett nombre d'occurrences:1
Balthy nombre d'occurrences:1
Albon nombre d'occurrences:1
homonymat nombre d'occurrences:1
mugichienne nombre d'occurrences:1
Simbad nombre d'occurrences:1
Gorringe nombre d'occurrences:1
Townshend nombre d'occurrences:1
Bassorah nombre d'occurrences:1
Amara nombre d'occurrences:1
el nombre d'occurrences:1
Kout nombre d'occurrences:2
bellums nombre d'occurrences:1
Euphrate nombre d'occurrences:1
mondanisee nombre d'occurrences:1
```

- Le test de la structure de la table de hachage

```
gcc -o bin/testdicoradix obj/dico_radix.o obj/testdicoradix.o -std=c99 -Wall -Wextra -g -pg -I./include -ln
gcc -c -std=c99 -Wall -Wextra -g -pg -I./include -ln test/verificateur_orthographique.c -o obj/verificateur_orthographique.o
gcc -o bin/verificateur_orthographique obj/verificateur_orthographique.o obj/lecture_dico_list.o obj/dico_list.o obj/lecture_dico_hashtable.o obj/dico_hashtable.o obj/lecture_dico_arbreprefixe.o obj/dico
arbreprefixe.o -std=c99 -Wall -Wextra -g -pg -I./include -ln
bamba@bamba-Latitude-7490:~/Documents/apman/tp_apman$ ./bin/test_lect_dico_hashtable FR.txt a_la_recherche_du_temps_perdu.txt
Le programme a trouvé 26490 erreurs:
refoulante nombre d'occurrences:1
Ivanovna nombre d'occurrences:3
pianola nombre d'occurrences:13
charentonnesque nombre d'occurrences:1
Montalivet nombre d'occurrences:5
decouchages nombre d'occurrences:1
Merowig nombre d'occurrences:1
Hunolstein nombre d'occurrences:1
Herodote nombre d'occurrences:1
Sausstier nombre d'occurrences:4
Duguay nombre d'occurrences:2
Lucullus nombre d'occurrences:2
menult nombre d'occurrences:1
Sandro nombre d'occurrences:2
Lebedev nombre d'occurrences:1
Meurles nombre d'occurrences:1
Guilbert nombre d'occurrences:1
aperit nombre d'occurrences:1
renonculaces nombre d'occurrences:1
Montesqueu nombre d'occurrences:4
Bulow nombre d'occurrences:1
antidreyfusards nombre d'occurrences:5
Albius nombre d'occurrences:1
Lemaitre nombre d'occurrences:3
MM nombre d'occurrences:4
San nombre d'occurrences:3
Aladin nombre d'occurrences:3
enollie nombre d'occurrences:1
Poucet nombre d'occurrences:1
poutana nombre d'occurrences:1
Apennins nombre d'occurrences:1
Sarrazine nombre d'occurrences:1
Dechambre nombre d'occurrences:16
hbarliste nombre d'occurrences:1
Tarquin nombre d'occurrences:2
Izraëltte nombre d'occurrences:1
aboutonnez nombre d'occurrences:1
Courgivaux nombre d'occurrences:1
Arthur nombre d'occurrences:2
Viareggio nombre d'occurrences:1
Noirmoutiers nombre d'occurrences:1
Cornely nombre d'occurrences:1
Brigode nombre d'occurrences:1
Lebas nombre d'occurrences:2
Lo nombre d'occurrences:1
Mantegna nombre d'occurrences:8
brac nombre d'occurrences:2
Loredan nombre d'occurrences:7
Theodhericulus nombre d'occurrences:1
```

● Le test de la structure de l’arbre préfixe

```
Mme
Villeparisis
Combray
Albertine
paperoles
Bloch
coplateurs
Bloch
paperoles
Combray
Norpois
Guernantes
Albertine
Rivebelle
Rivebelle
Albertine
Albertine
Hugo
II
Elysees
Caspienne
Mme
Mme
Sazerat
Mme
Mme
Sazerat
Mme
Mme
Sazerat
Infrequente
Bergotte
Sherlar
Elstir
Chardin
Albertine
Swann
Combray
Combray
Albertine
redessiner
Guernantes
Swann
Swann
Guernantes
Albertine
Combray
Guernantes

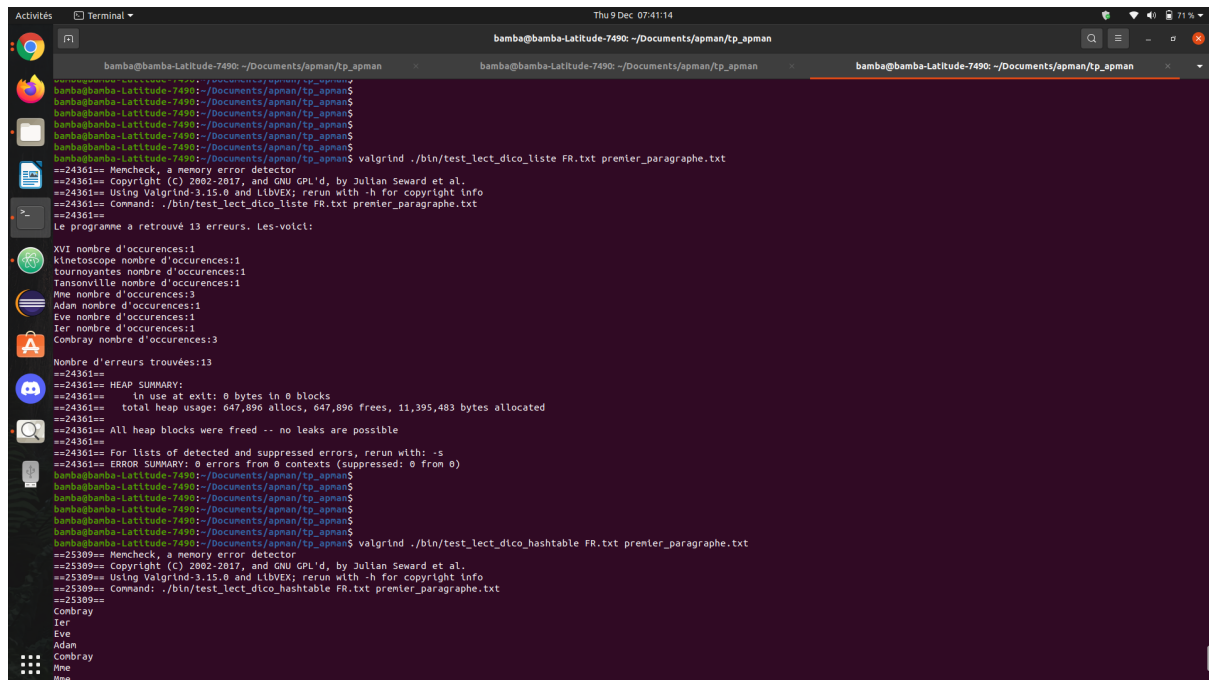
Ceci est le test avec l'arbre prefixe
Il y'a 26490 erreurs
bamba@bamba-Latitude-7490:~/Documents/apman/tp_apman$
```

IV°) Analyse comparatives des performances CPU et mémoire

Voici les données renvoyées par l’outil Valgrind concernant la bonne gestion de la mémoire.

- Pour la liste chaînée

Pour celle-ci on a appelée la fonction sur le premier paragraphe du texte par soucis d'optimisation et en comparaison avec l'appel de la fonction de hachage sur le même paragraphe: celle de la structure chaînée prend 11 395 483 octets de mémoire alors que la table de hachage en prend 16 497 631.

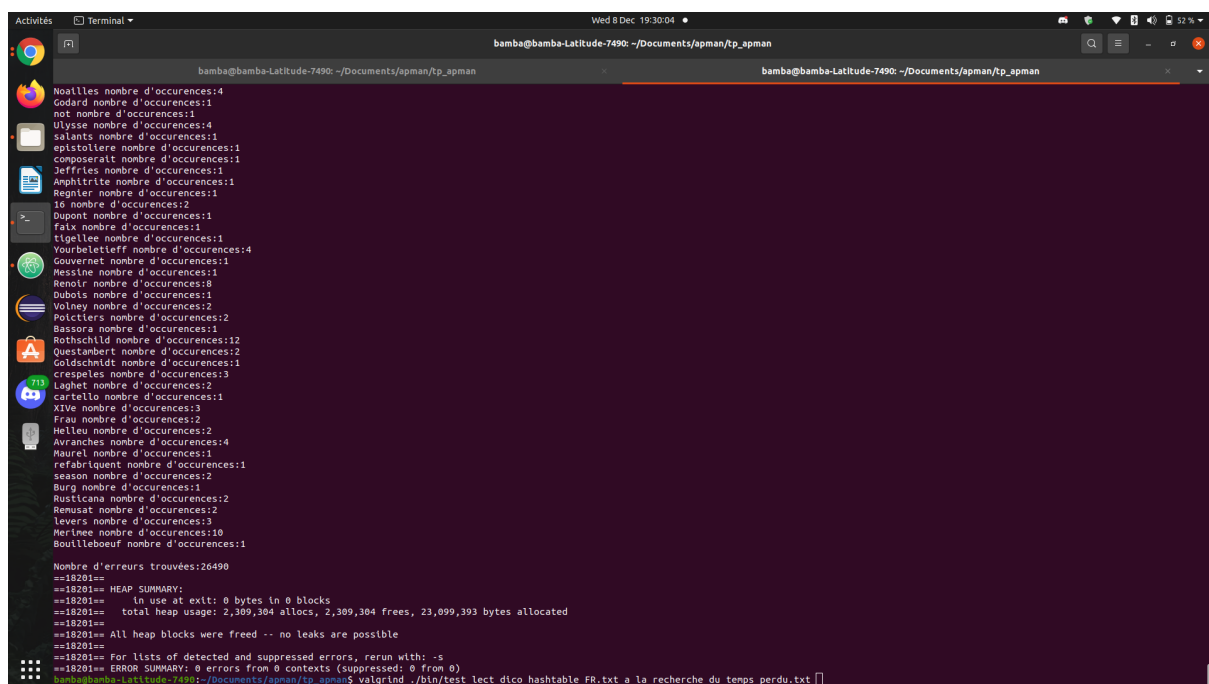


```
bamba@bamba-Latitude-7490: ~/Documents/apman/tp_apman
bamba@bamba-Latitude-7490: ~/Documents/apman/tp_apman$ valgrind ./bin/test_lect_dico_liste FR.txt prenier_paragraphe.txt
==24361== Memcheck, a memory error detector
==24361== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==24361== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==24361== Command: ./bin/test_lect_dico_liste FR.txt prenier_paragraphe.txt
==24361==
Le programme a retrouvé 13 erreurs. Les voici:

XVI nombre d'occurrences:1
kinetoscope nombre d'occurrences:1
tournoyantes nombre d'occurrences:1
Tansonville nombre d'occurrences:1
Mme nombre d'occurrences:3
Adam nombre d'occurrences:1
Eve nombre d'occurrences:1
Ier nombre d'occurrences:1
Conbray nombre d'occurrences:3

Nombre d'erreurs trouvées:13
==24361==
==24361== HEAP SUMMARY:
==24361==   in use at exit: 0 bytes in 0 blocks
==24361==   total heap usage: 647,896 allocs, 647,896 frees, 11,395,483 bytes allocated
==24361==
==24361== All heap blocks were freed -- no leaks are possible
==24361==
==24361== For lists of detected and suppressed errors, rerun with: -s
==24361== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
bamba@bamba-Latitude-7490: ~/Documents/apman/tp_apman$
```

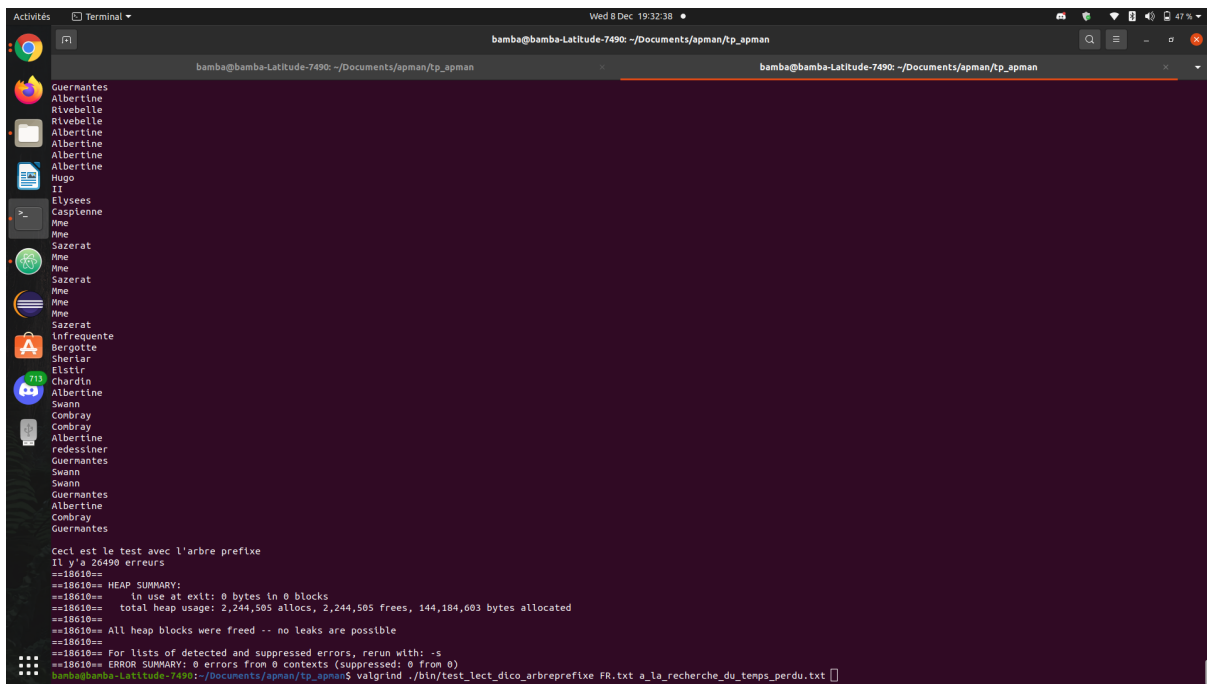
- Pour la table de hachage:



```
bamba@bamba-Latitude-7490: ~/Documents/apman/tp_apman
bamba@bamba-Latitude-7490: ~/Documents/apman/tp_apman$ valgrind ./bin/test_lect_dico_hashtable FR.txt a_la_recherche_du_temps_perdu.txt
==18201== Memcheck, a memory error detector
==18201== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==18201== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==18201== Command: ./bin/test_lect_dico_hashtable FR.txt a_la_recherche_du_temps_perdu.txt
==18201==
Noailles nombre d'occurrences:4
Godard nombre d'occurrences:1
not nombre d'occurrences:1
Ulysse nombre d'occurrences:4
salanis nombre d'occurrences:1
epistolere nombre d'occurrences:1
composeraIt nombre d'occurrences:1
Jeffries nombre d'occurrences:1
Amphitrite nombre d'occurrences:1
Regnier nombre d'occurrences:1
Id nombre d'occurrences:2
Dupont nombre d'occurrences:1
faix nombre d'occurrences:1
tigelle nombre d'occurrences:1
Yourbiestieff nombre d'occurrences:4
Gouvernet nombre d'occurrences:1
Messine nombre d'occurrences:1
Renot nombre d'occurrences:10
Dubois nombre d'occurrences:1
Volney nombre d'occurrences:2
Polctters nombre d'occurrences:2
Bassora nombre d'occurrences:1
Rothschld nombre d'occurrences:12
Questambert nombre d'occurrences:2
Goldschmidt nombre d'occurrences:11
crespeles nombre d'occurrences:3
Laghet nombre d'occurrences:2
cartello nombre d'occurrences:1
Xive nombre d'occurrences:3
Frau nombre d'occurrences:2
Helieu nombre d'occurrences:2
Avranches nombre d'occurrences:4
Maurel nombre d'occurrences:1
refabrique nombre d'occurrences:1
season nombre d'occurrences:2
Burg nombre d'occurrences:1
Rusticana nombre d'occurrences:2
Remusat nombre d'occurrences:2
levers nombre d'occurrences:3
Mernee nombre d'occurrences:10
Bouilleboeuf nombre d'occurrences:1

Nombre d'erreurs trouvées:26490
==18201==
==18201== HEAP SUMMARY:
==18201==   in use at exit: 0 bytes in 0 blocks
==18201==   total heap usage: 2,309,304 allocs, 2,309,304 frees, 23,099,393 bytes allocated
==18201==
==18201== All heap blocks were freed -- no leaks are possible
==18201==
==18201== For lists of detected and suppressed errors, rerun with: -s
==18201== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
bamba@bamba-Latitude-7490: ~/Documents/apman/tp_apman$
```

- Pour l'arbre préfixe



The screenshot shows a terminal window with a dark background. The title bar indicates the user is 'bamba' on a machine named 'bamba-Latitude-7490'. The terminal displays the output of a program, which includes a list of names on the left side of the screen and a series of memory-related statistics on the right. The statistics show that the program used 2,244,505 bytes of memory, with 144,184,603 bytes allocated and 23,099,393 bytes freed. The output also mentions that there were no leaks and that the program ran successfully.

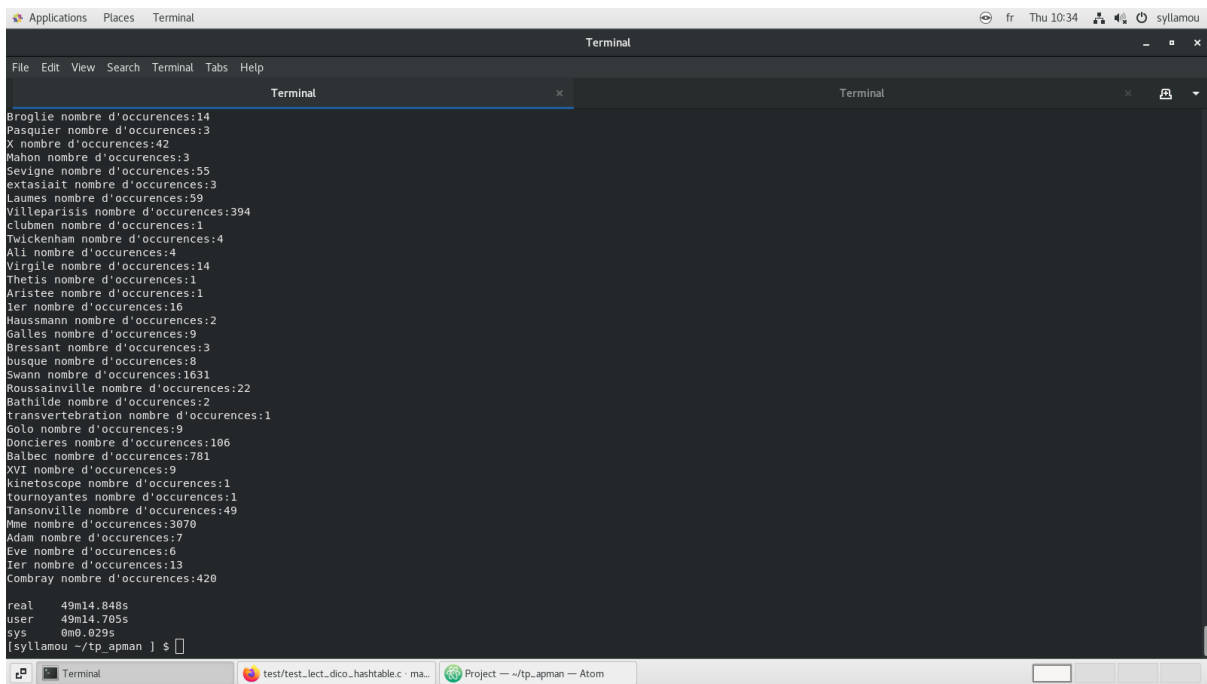
```
Guernantes
Albertine
Rivebelle
Rivebelle
Albertine
Albertine
Albertine
Hugo
II
Elysees
Caspienne
Mme
Mme
Sazerat
Mme
Sazerat
Mme
Mme
Sazerat
infrequente
Bergotte
Charlar
Elstir
Chardin
Albertine
Swann
Combray
Combray
Albertine
redessiner
Guernantes
Swann
Swann
Guernantes
Albertine
Combray
Guernantes

Ceci est le test avec l'arbre préfixe
Il y'a 26490 erreurs
==18010==
==18010== HEAP SUMMARY:
==18010==    in use at exit: 0 bytes in 0 blocks
==18010==   total heap usage: 2,244,505 allocs, 2,244,505 frees, 144,184,603 bytes allocated
==18010==
==18010== All heap blocks were freed -- no leaks are possible
==18010==
==18010== For lists of detected and suppressed errors, rerun with: -s
==18010== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
bamba@bamba-Latitude-7490: ~/Documents/apman/tp_apman$ valgrind ./bin/test_lect_dico_arbreprefixe FR.txt a_la_recherche_du_temps_perdu.txt
```

On remarque aussi à travers ces données que l'arbre de préfixe occupe environ 7 fois plus de mémoire que la table de hachage (144 184 603 bytes contre 23 099 393 bytes).

En utilisant la commande time, voici le temps que prend nos différentes structures:

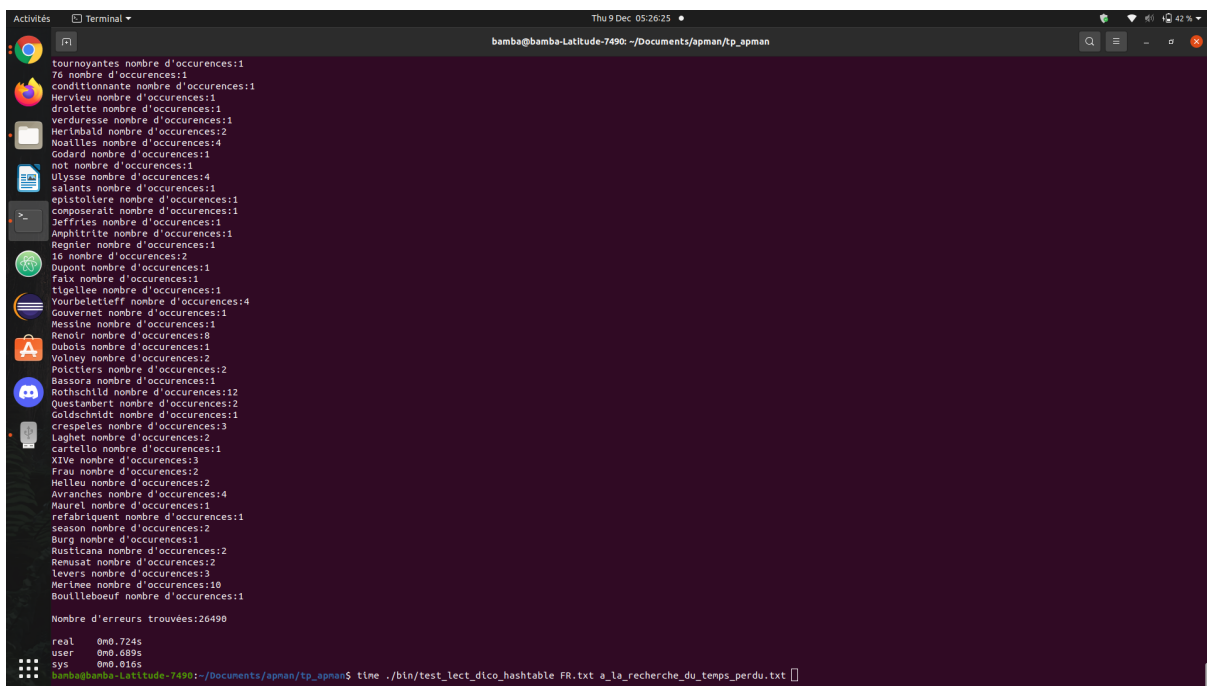
- Liste chaînée



```
Applications  Places  Terminal
Terminal
File Edit View Search Terminal Tabs Help
Terminal
Brogie nombre d'occurrences:14
Pasquier nombre d'occurrences:3
X nombre d'occurrences:42
Mahon nombre d'occurrences:3
Sevigne nombre d'occurrences:55
extasiait nombre d'occurrences:3
Leumes nombre d'occurrences:59
Villeparisis nombre d'occurrences:394
clubmen nombre d'occurrences:1
Twickenham nombre d'occurrences:4
Ali nombre d'occurrences:4
Virgile nombre d'occurrences:14
Thetis nombre d'occurrences:1
Aristee nombre d'occurrences:1
Ier nombre d'occurrences:16
Hausmann nombre d'occurrences:2
Galles nombre d'occurrences:9
Bressant nombre d'occurrences:3
busque nombre d'occurrences:8
Swann nombre d'occurrences:1631
Roussainville nombre d'occurrences:22
Bathilde nombre d'occurrences:2
transvertebration nombre d'occurrences:1
Golo nombre d'occurrences:9
Doncieres nombre d'occurrences:106
Balbec nombre d'occurrences:781
XVI nombre d'occurrences:9
kinetoscope nombre d'occurrences:1
tourneyantes nombre d'occurrences:1
Tansonville nombre d'occurrences:49
Mme nombre d'occurrences:3070
Adam nombre d'occurrences:7
Eve nombre d'occurrences:6
Ier nombre d'occurrences:13
Combray nombre d'occurrences:420

real    49m14.848s
user    49m14.705s
sys     0m0.029s
[syllamou ~/tp_apman ] $
```

- Table de hachage

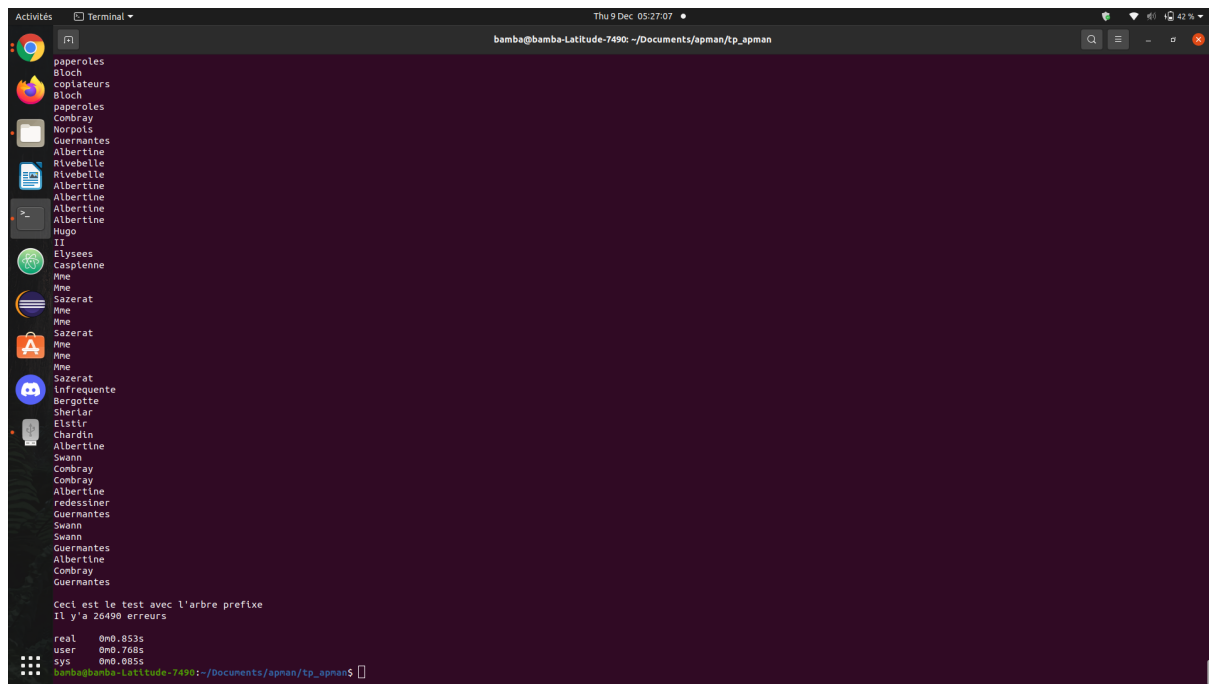


```
Activités  Terminal
Thu 9 Dec 05:26:25
bamba@bamba-Latitude-7490: ~/Documents/apman/tp_apman
tourneyantes nombre d'occurrences:1
76 nombre d'occurrences:1
conditionnante nombre d'occurrences:1
Hervieu nombre d'occurrences:1
drolette nombre d'occurrences:1
verdresse nombre d'occurrences:1
Herinbald nombre d'occurrences:2
Noailles nombre d'occurrences:4
Godard nombre d'occurrences:1
not nombre d'occurrences:1
Ulysse nombre d'occurrences:4
salants nombre d'occurrences:1
epistolere nombre d'occurrences:1
composerait nombre d'occurrences:1
Jeffries nombre d'occurrences:1
Amphitrite nombre d'occurrences:1
Regnier nombre d'occurrences:1
16 nombre d'occurrences:2
Dupont nombre d'occurrences:1
faix nombre d'occurrences:1
tigellee nombre d'occurrences:1
Vourbeletieff nombre d'occurrences:4
Gouvernet nombre d'occurrences:1
Messine nombre d'occurrences:1
Renolr nombre d'occurrences:8
Dubois nombre d'occurrences:1
Volney nombre d'occurrences:2
Polctiers nombre d'occurrences:2
Bassora nombre d'occurrences:1
Rothschild nombre d'occurrences:12
Questambert nombre d'occurrences:2
Goldschmidt nombre d'occurrences:1
crespeles nombre d'occurrences:3
Laghet nombre d'occurrences:2
cartello nombre d'occurrences:1
Xlve nombre d'occurrences:3
Frau nombre d'occurrences:2
Helleu nombre d'occurrences:2
Avranches nombre d'occurrences:4
Maurel nombre d'occurrences:1
refabriquient nombre d'occurrences:1
season nombre d'occurrences:2
Burg nombre d'occurrences:1
Rusticana nombre d'occurrences:2
Renusat nombre d'occurrences:2
Levers nombre d'occurrences:3
Meritee nombre d'occurrences:10
Bouilleboeuf nombre d'occurrences:1

Nombre d'erreurs trouvées:26490

real    0m0.724s
user    0m0.689s
sys     0m0.016s
bamba@bamba-Latitude-7490: ~/Documents/apman/tp_apman$ tline ./bin/test_lect_dico_hashtable FR.txt a_la_recherche_du_temps_perdu.txt
```

- Arbre préfixe



```
paperoles
Bloch
copiateurs
Bloch
paperoles
Combray
Norpois
Guernantes
Albertine
Rivebelle
Rivebelle
Albertine
Albertine
Albertine
Hugo
II
Elysees
Caspienne
Mme
Mme
Sazerat
Mme
Sazerat
Mme
Mme
Sazerat
Infrequente
Bergotte
Sheriar
Elstir
Chardin
Albertine
Swann
Combray
Combray
Albertine
redessiner
Guernantes
Swann
Swann
Guernantes
Albertine
Combray
Guernantes

Ceci est le test avec l'arbre prefixe
Il y'a 26490 erreurs

real    0m0.853s
user    0m0.768s
sys     0m0.085s
bamba@bamba-Latitude-7490: ~/Documents/apman/tp_apman$
```

Ces résultats sont cohérents et témoignent de la complexité de nos fonctions. En effet, dans la liste chaînée, toute recherche se fait en temps linéaire $O(n)$. Si on néglige les autres opérations (lecture du fichier texte et création du dictionnaire qui restent coûteuses mais le sont moins que la recherche), on se retrouve avec une complexité moyenne de $O(N*N')$ avec N qui est le nombre de mots dans le texte de vérification et N' le nombre de mots dans le dictionnaire. On se retrouve donc dans la structure de liste chaînée avec une complexité temporelle d'au moins quadratique $O(N^2)$ à la taille du dictionnaire qui est énorme (plus de 300000 mots). La structure liste n'est pas adéquate à ce type d'utilisation, la complexité spatiale n'est pas énorme mais celle temporelle est trop grande pour des fichiers de cette taille.

La structure table de hachage, pour la recherche et l'insertion, à une complexité de $O(1+n/k)$ avec n le nombre de mots dans le dictionnaire et k la taille de la table de hachage. Étant donné que l'on a utilisé une bonne fonction de hachage et que la taille choisie est importante ($\frac{1}{3}$ de la taille du dictionnaire), notre complexité n'est pas linéaire et la plupart des opérations se fait en temps constant même si à cause des collisions certaines se font en temps linéaire. En prenant en compte l'ensemble des opérations lors du test, on peut dire que cette structure a pour complexité globale $O(N)$ avec N le nombre de mots à vérifier.

Dans la structure de l'arbre préfixe, la recherche et l'insertion de tout mot se fait en $O(k)$ avec k la longueur du mot. On est ainsi assurés que ces opérations se font tout le temps en temps constant (là où dans la table de hachage certaines de ces opérations sont linéaires). La complexité globale de notre fonction peut donc être prise $O(N)$ avec N nombre de mots à vérifier. En termes de complexité temporelle, l'arbre préfixe est légèrement plus efficace que la table de hachage (dans les captures ajoutées à ce rapport, la table de hachage prend moins de temps grâce à la manière dont on traite les erreurs et les bons choix effectués sur la table de hachage). Mais en termes de mémoire occupée, l'arbre préfixe devient tout de suite moins intéressant. On peut, en effet, considérer que la table de hachage a, comme la liste chaînée, une complexité spatiale de $O(n)$ -la table de hachage occupe une plus grande mémoire que la liste mais la différence n'est pas énorme au vu des choix effectués sur la fonction de hachage et la taille-. Ceci en ne rentrant pas dans les détails quant à la taille qu'occupent les mots en tant que tels mais juste du nombre de mots à stocker. L'arbre préfixe quant à lui utilise des nœuds et chaque nœud correspond non pas à un mot mais à une lettre d'un mot. Ainsi donc en terme d'espace, on est en moyenne à $O(N*k)$ avec k le nombre moyen de lettres dans un mot et N le nombre de mots à stocker. Ceci est toujours linéaire mais reste plus important que pour la table de hachage ce qui explique les résultats fournis par le Valgrind le nombre de d'octets utilisé étant 7 fois plus important.

Ainsi donc pour résumer, la liste chaînée est la structure qui utilise le moins la mémoire mais c'est la moins efficace en termes de complexité temporelle qui est quadratique. La table de hachage, quant à elle, occupe un peu plus d'espace que la liste chaînée mais a une bien meilleure complexité temporelle avec un bon temps d'exécution. L'arbre préfixe a lui aussi une bonne complexité temporelle et est la plus rapide de nos 3 structures. Toutefois, elle utilise une quantité de mémoire bien plus importante. Si l'on doit effectuer un compromis niveau temps d'utilisation et mémoire, la table de hachage est la structure qu'il faut utiliser. Si en revanche, on souhaite avoir un temps d'exécution le plus petit possible sans faire attention à la mémoire, il faudra recourir à l'arbre préfixe. Si toutefois, la taille des données à traiter n'est pas énorme et qu'on souhaite utiliser le moins de mémoire possible, la liste chaînée est particulièrement adaptée mais elle ne l'est pas sur de gros fichiers tels que des dictionnaires.